Docket No. 42P17982
Express Mail No. EV339914055

UNITED STATES PATENT APPLICATION

FOR

METHOD AND MECHANISM FOR PROGRAMMABLE FILTERING OF TEXTURE MAP DATA IN 3D GRAPHICS SUBSYSTEMS

Inventor:

Kim Pallister

Prepared By:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 Wilshire Boulevard, 7th Floor
Los Angeles, California 90025-1030
(310) 207-3800

METHOD AND MECHANISM FOR PROGRAMMABLE FILTERING OF TEXTURE MAP DATA IN 3D GRAPHICS SUBSYSTEMS

BACKGROUND

Field of the Invention

[0001] The embodiments of the invention relate to computer graphics. More specifically, embodiments of the invention relate to processing of texture map data.

Background

[0002] Graphics applications, and particularly three dimensionally (3D) graphic applications have long been one of the most processing intensive activities performed by personal computers. To improve graphic processing capabilities, graphics co-processors have proliferated and are widely available on most modern day personal computers. Graphic coprocessors are specialized integrated circuits designed to quickly perform processing intensive tasks required by graphic applications.

[0003] The transformation of scene information (source data) into 3D images (display output) requires a number of operations. These operations in aggregate are referred to as a 3D graphics rendering pipeline. The operations performed by the pipeline can be grouped into certain fundamental functionalities. One of these functionalities is texture mapping. Texture mapping is a process in which the one, two or three dimensional image representing an object surface properties (such as appearance, reflectivity, or other such properties) is applied to a three dimensional mesh representing the object in a final rendering. While a two dimensional image is most commonly used, other dimensionalities are possible.

[0004] It is frequently the case when a texture image is applied to an object in a final rendering, there is disparity between a number of sample texture elements (texels) and the source texture image and the number of picture elements (pixels) to which the image is mapped. When the number of texels in a given range is less than the number of pixels, then the texture is required to be upsampled. When upsampling a texture, a scheme must be used to fill intermediate values. This scheme is referred to herein as "texture filtering" and has largely been performed by a fixed function state machine. Most current graphic coprocessor support four types of texture filtering; point sampling, bilinear filtering, trilinear filtering and anisotropic filtering. As the

filtering methods become increasingly complex, the state machine required to perform them becomes increasingly complex and requires increased real estate within the graphics coprocessor. This coupled with the fact that uses for texture data continues to expand, for example, texture data is being used for lighting and other surface properties in addition to color, renders the commonly employed linear interpolation inefficient or even insufficient.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] Embodiments of the invention are illustrated by way of example and not by way of limitation in the figures of the accompanying drawings in which like references indicate similar elements. It should be noted that different references to "an" or "one" embodiment in this disclosure are not necessarily to the same embodiment, and such references mean at least one.

[0006] **Figure 1A** is a block diagram of a system of one embodiment of the invention.

[0007] **Figure 1B** is a diagram of texture sampling in one embodiment of the invention.

[0008] **Figure 2** is a flow diagram of the setup of the textured filtering module in one embodiment of the invention.

[0009] **Figure 3** is a flow diagram of texture filtering in one embodiment of the invention.

DETAILED DESCRIPTION

[0010] **Figure 1A** is a block diagram of a system of one embodiment of the invention. A host processor 100 is coupled by a bus 102 to a memory 104. A graphics coprocessor 106 is also coupled to the bus 102. Additionally, graphics coprocessor 106 may be coupled to memory 104 by an accelerated graphics port (AGP) 112. The AGP may adhere to Accelerated Graphics Port AGP V3.0 Interface Specification Rev. 1.0 published September 2002 (hereinafter the AGP Specification). AGP 112 allows rapid access to graphics data residing in memory 104. Also coupled to the bus are framebuffer 108 and display 110. In some embodiments, framebuffer 108 may be contained within memory 104. Graphics coprocessor 106 includes pixel processing pipeline 120. Within the pixel processing pipeline 120 is a vertex processing module 122, primitive assembly module 124, a fragment processing module 126 and a framebuffer processing module 128. Vertex processing module 122 in operation receives vertex data, which may include, for example, 3D

positional information, color information and other similar information related to vertices in the graphic image. In one embodiment, vertex data is of the form V = X, Y, Z, T_u, T_v, RGB. In this expression, X, Y, Z are the three dimensional cartesian coordinates of the vertex, T_u and T_v are the two dimensional coordinate of the corresponding texel in the texture map and RGB are the red, green and blue color values at the vertex. Other forms and contents of vertex data are also contemplated.

[0011] Vertex processing module does three-dimensional transformations on 3D positional data conveyed, and may, for example, applies lighting. The processed vertices are passed to the primitive assembly module, which receives connectivity data. The connectivity data may include indices to permit assembly of primitives, typically triangles, based on the vertices and indices received.

[0012] The primitives are passed to the fragment processing module 126 which processes the primitives to identify fragments and apply texture data to build an output. As used herein, "fragment" refers to a pixel or group of contiguous pixels that are to be consistently processed to generate the output. The fragment processing exchanges data relating to texture mapping the fragments with a texture filter module 130.

[0013] The texture filter module 130 communicates with fragment processing module 126 to supply texels for application to the pixels. In one embodiment, texture filter module 130 is programmable. In this context, programmable is deemed to mean capable of executing a software program consisting of one or more instructions from a defined instruction set. One example of an instruction set is set forth below in Table 1.

Instruction	Description
ADD A,B	Adds A and B operands
SUB A,B	Subtracts B from A
MUL A,B	Multiples A by B
RCP A,B	Makes A the reciprocal of B
CMP A,B, X	Compares A, B according to immediate X, places result in A
MIN A,B	Compares A, B leaves minimum of two values in A
MAX A,B	Compares A, B leaves maximum of two values in A
MOV A,B	Moves B into A

TABLE 1

[0014] Alternative instruction sets, either shorter or longer, may be employed in various embodiments of the invention.

[0015] In one embodiment, texture filter module 130 includes a plurality of texture processing cores (TPCs) 132 (16 TPC are shown in Figure 1). Other embodiments may have more or fewer TPCs. In one embodiment, a single TPC exists. In one embodiment, each TPC 132 is capable of processing a pixel in parallel with each of the other TPC 132. Each core 132 may be provided with a register set 134 which may include various types of registers such as control registers, source registers, temporary registers and an output register.

[0016] In one embodiment, the control registers include a sampling register, a status register, an address register, an offset register, and a plurality of fraction registers. In one embodiment, the sampling register has one bit corresponding to each source registers indicating whether the source register should be sampled or not. For example, if there are sixteen source registers, the sampling register may be a sixteen bit register with one bit corresponding to each of the sixteen source registers. In one embodiment, the status register is used to indicate the status of the TPC after certain conditions, such as overflow, divide by zero, etc. In one embodiment, the address register may be a 32 bit register containing the address of the texture map data. In one embodiment, this register may be accessible only by an application programming interface (API) rather than providing direct access to a programmer. The offset register may be used to provide an offset into the

texture data corresponding to the nearest texel coordinate. Fraction registers may be used to hold the fractional coordinate between the texel samples in each dimensionality. In one embodiment, these would be provided by the fragment processing module 126. In one embodiment above, where V = X, Y, Z, T_u, T_v, RGB; T_u and T_v correspond to a pixel to be texture mapped would be provided to the texture filtering module. As one example, an eight pixel one dimensional texture coordinate of 0.175 would fall between the second (0.125) and third (0.25) texel. It would equate to a fraction of .2. The fraction in this embodiment is found as (.175 - .125)/.125 or more generally, the coordinate less the closest lower increment divided by the increment value.

[0017] **Figure 1B** is a diagram of texture sampling in one embodiment of the invention. In one embodiment, sixteen source registers are provided. With each register corresponding to one texel in a 4 X 4 grid surrounding the T_uT_v sampling location of the texture sample point and would correspond to pixels addressed in such a fashion. While T_uT_v mapps to a location between texels 5 and 6 and texels 9 and 10, the contribution of the sixteen texels in the patch to the texture value assigned to T_uT_v may be defined by the texture filtering program. In some embodiments, only texels 5, 6, 9 and 10 provide a contribution. In other embodiments, all sixteen texels may contribute. In still other embodiments, all diagonal prixels in the group may contribute. As illustrated, the programmable nature of the texture filtering module permits robust and flexible texture filtering options.

[0018] Temporary registers may be provided for optional use by a programmer performing intermediate calculations on sample data. An output register is provided to store the output once the filtering operation is complete. In one embodiment, a 32 bit register is provided to receive the final result. Larger registers may be employed, however, in some embodiments a 32 bit ARGB (alpha red green blue) value is deemed sufficient.

[0019] The actual filtering may be performed by the texture filtering module 130 by loading a desired filtering program into a textured processing core. The filtering program corresponds to a fragment to be processed. Within a region of an image, it may be desirable to apply various effects to the texture data accordingly. Thus, for a particular graphic image, there may be numerous filtering programs employed. For example, the filter program applied to a shiny part of a leather jacket on an image would likely to be

different than the filter program applied to a scuffed part of a leather jacket. By using different programs in e texture filter module, the different effect can be accommodated. The usage of several filter programs during the course of rendering a given scene image is analogous to how, under the current-day fixed-function schemes, the rendering of a given scene may involve switching between the different fixed-function filtering states for different parts of the scene.

[0020] The program employed will influence which of the e.g. 16 texels are actually sampled to perform the texture filtering . In one embodiment, texture data may be arranged in memory to optimize access to the texels likely to be sampled. For example, if the sampling register indicates every fourth texel value is active, the texture data may be stored so that points 1, 5, 9 and 13 are contiguous in memory, points 2, 6, 10, etc. are contiguous. As another example, where every second texel is active, 1, 3, 5, 7, etc. are contiguous and 2, 4, 8, etc. are contiguous. This arrangement in memory may be performed by the host processor 100 or the graphic coprocessor 106.

[0021] Arranging memory requires a certain amount of processor resources, in one embodiment, a determination is made when the likely use of the texture data exceeds a threshold cost required to rearrange it. Thus, where the usage of the textured data justifies the cost to rearrange it, in one embodiment, the textured data is rearranged in memory to facilitate access. The threshold may be selected based on objective guidelines such as number of texels to be processed with a given program.

[0022] Once texture filtering is complete and the output generated, the output value may then be passed back to fragment processing module 126 to permit the output fragment to be built. The output built by the fragment processing module 126 is passed to framebuffer processing module 128. Framebuffer processing module 128 combines the pixels received with the existing framebuffer for output to display 110.

[0023] **Figure 2** is a flow diagram of the setup of the textured filtering module in one embodiment of the invention. At functional block 202, a texture filter program is loaded into a texture processing core of the texture filtering module. At functional block 204, the sampling register is initialized to indicate the texels surrounding a sample point that will be sampled as part of the filtering process. At decision block 206, a determination is made if the

texels to be sampled in conjunction with the number of samplings required justify reorganization of the texture data in memory. If so, the texture data is reordered for efficient access at functional block 208. After reordering, or if no reordering is required, the address register is initialized at functional block 210.

[0024] **Figure 3** is a flow diagram of texture filtering in one embodiment of the invention. At functional block 302, the texture filter fetches the coordinate data for pixels to be rendered from the vertex pipeline. In one embodiment, these will be fetched from the fragment processing module. At functional block 304, texel values identified from the sampling register are fetched from memory. At functional block 306, fraction registers are loaded with the coordinate data. The texture processing cores are used to execute a filter program at functional block 308. At decision block 310, a determination is made if the execution of the filter program necessitates setting of a status flag. If the execution requires the status flag such as a divide by zero or overflow, the status register is loaded with an appropriate value at functional block 312. If no status flag is required, or after the status register has been loaded, the output value is loaded into the output register and the output is signaled to be available at functional block 314. At decision block 316, a determination is made if there are more pixels to be rendered using the existing filter program. If so, a flow continues. If not, it ends.

[0025] Although the above flow diagrams are arranged in a particular order, it should be understood that some of the operations may be performed in parallel or in a different order than depicted. Accordingly, such parallization or rearrangement is within the scope and contemplation of the embodiments of the invention. It should also be noted that while in only one embodiment, a single texture processing core may be present in the texture filter module, embodiments with multiple texture processing cores, pixels may be processed in parallel with each core following the flow depicted in Figure 3.

[0026] In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes can be made thereto without departing from the broader spirit and scope of the invention as set forth in the

appended claims. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.